Collaborative Writing of XML Documents

Hala Skaf-Molli, Pascal Molli, Charbel Rahhal, Hala Naja-Jazzar

Collaborative Writing of XML Documents

Hala Skaf-Molli, Pascal Molli, and Charbel Rahhal
LORIA – INRIA Lorraine,
Université de Nancy, France
Email: skaf@loria.fr, molli@loria.fr, rahalcha@loria.fr

Hala Naja-Jazzar
Faculty of Science - Branch 3
Lebanese University, Lebanon
Email: hjazzar@ul.edu.lb

***Abstract*—Collaborative writing is the process of two or more people working together to create a common document. People can be distributed in time, in place and across organizations. They can share writing different kinds of documents. In this paper, we focus on collaborative writing of XML documents. XML documents must be validated via a set of constraints called DTD to be considered as consistent. In cooperative writing, shared XML documents are replicated on different sites. This improves the availability of the documents. The modifications of a replica at a site are sent and integrated on all the others sites, in order to ensure the convergence of the replicas. While on each site, a replica can be validated via the DTD. After the reconciliation (also known as merging) of the different replicas, the result of merging consistent replicas may not validate the DTD. A major problem of this result, that is not possible to open a document with the tool used to edit it. To overcome this problem, we propose an automatic approach to repair inconsistency.**

KEY WORDS:

CSCW, XML document, DTD validation, automatic repairing.

I. INTRODUCTION

Collaborative writing (CW) is the process of two or more people working together to create a common document. People involved in CW can be distributed in time, in place and across organizations. Collaborative writing is standard practice in technical and scientific settings; some examples include research papers, software development, proposals for funding and user manuals.

When collaboration is efficiently managed, the advantages of working in a group include increased efficiency, reduced errors and the benefits of different viewpoints and expertise [17], [11]. If collaboration is poorly supported, it can lead to inconsistencies, misunderstandings, conflicts, redundant work and coordination problems.

In this work, we are interested in collaborative writing of *XML* (eXtensible Markup Language) documents. *XML* document usually comes with a Document Type Definition *DTD* that specifies the structure of the document by defining the legal building blocks of an *XML* document. For instance, a DTD can specify that an XML document must have only one title. This is a kind of structural constraints on the *XML* document. An *XML* document is valid (or consistent) if it conforms to its *DTD*. Traditionally, XML-based format environment has a built-in function to verify the validation of the *XML* document. In case of a violation of the *DTD*, an error message is displayed and the user has to modify her document else she cannot save it. This means that the author can produce only valid documents.

In collaborative writing, when several authors work together to edit a document. The shared document is usually replicated at the site of each participant [8]. This allows more availability of the document, in addition, each participant has her own replica of the document, she can work insulated in her workspace. This phase of cooperation produces copies divergence [3], [8]. From times to times copies have to be resynchronized in order to integrate the contribution of the different participants.

There are several algorithms to synchronize *XML*, such as SO6 [12] and DeltaXML [1]. However, existing algorithms do not ensure the validation of the *DTD*. Therefore, the merging phase can produce a document that does not validate the *DTD*, even each different replica is valid. Thus, it is not possible to open the document with the tool that generated it. This constitutes a major problem.

A possible solution is to use the approach proposed by IceCube [6]. IceCube tries to find an ordering (Scheduling) of concurrent operations that respects constraints. An operation that does not respect the constraint is canceled .The major drawbacks of IceCube are the risk of combinatorial explosion, the annulation of some operations and finally how to transform *DTD* to IceCube constraints. Annulation of operations is not an acceptable solution in collaborative writing.

In order to develop an environment that respects the requirements of collaborative writing of *XML* and the validation of the *DTD*. We propose an automatic approach to repair this violation.

If the result of merging different remote modifications is a corrupted document, a set of repairing actions is proposed to re-establish the consistency of the document. For example, if after the merge, the document has two titles instead of one title (according to its *DTD*). Possible repairing actions are: (1) delete one of the two titles, (2) put one of the titles as a comment, and finally (3) concatenate the two titles into one title.

Generally, it is possible to repair a document in many different ways. However, in collaborative context, repairing actions must respect the intention of the user and must form a minimal set of changes to the original document. This means that the work done by a user must not disappear after repairing the inconsistency. A person who participates in collaborative writing will disagree to lose her work because

of the "inconsistency" problem.

In previous work, we described how the operational transformation approach (OT) was used as a theoretical foundation to build a generic and safe synchronizer [7]. Later, we defined the specific transformation functions to synchronize XML documents [12]. These functions are integrated in SO6 tool. SO6 ensures convergence of the replicas but it does not ensure the validation of the DTD. It is well known that the operational transformation approach maintains syntactic consistency of replicated data but provide no guarantee on semantic consistency [16], [14]. XML documents can be corrupted after reconciliation. In this paper, we propose a repairing approach in order to overcome this problem.

The paper is structured as follows. Section II introduces the operational transformation approach which serves as a theoretical foundation for XML reconciliation tool SO6. Section III presents a scenario of collaborative writing of XML documents. This scenario shows the result of the application of SO6 and motivates our work. Section IV explains the repairing approach and shows how we can apply existing repairing algorithms in collaborative writing of XML documents. Section V concludes and points out some limits of our works and future work.

II. BACKGROUND

This section describes the Operational Transformation approach (OT) that is the theoretical foundation of the generic and safe synchronizer. OT [4] is an optimistic replication model used in real-time group editors domain. OT considers n sites, each site owns a copy of shared data. When a site performs an update, it generates a corresponding operation, which is first executed locally and then broadcasted to other sites. Every operation is processed in four steps: (a) generated on one site, (b) broadcasted to other sites, (c) received by other sites, (d) executed on other sites.

The execution context of a received operation op_i may be different from its generation context. In this case, the integration of op_i by other sites may lead to inconsistencies between replicas. For instance, we consider two sites $site_1$ and $site_2$ working on a shared data of type *string of characters* initially equal to the string "efect". A string of characters can be modified with the operation *ins(p,c)* for inserting a character c at position p in the string. We assume the position of the first character in a string is 0. $user_1$ and $user_2$ generate and execute two concurrent operations op_1=*ins(2,f)* and op_2=*ins(5,s)*, respectively. When op_1 is received and executed on $site_2$, it produces the expected string "effects". But, when op_2 is received on $site_1$, since it does not take into account that op_1 has been executed before it, its execution leads to the state "effecst". Finally, the copies of $site_1$ and $site_2$ do not converge.

In the operational transformation (OT) approach, before being executed, received operations are transformed regarding concurrent operations that were already executed on the local copy. This transformation is performed by calling transformation functions.

Definition A transformation function T takes two concurrent operations, op_1 and op_2, must be defined on a same state S. The function computes a new operation op'_1 equivalent to op_1 – i.e. has the same effects – but defined on the state $S' = S \odot op_2$. S' is the state resulting from the execution of op_2 on state S.

Using OT approach, our previous example is now executed as follows. When op_2 is received on $site_1$, op_2 needs to be transformed regarding op_1. The integration algorithm calls the transformation function $T(op_2$=*ins(5,s)*,op_1=*ins(2,f))* = *ins(6,s)* = op'_2. The insertion position of op_2 is incremented since op_1 has inserted an f before s in state "efect". After the execution of op'_2, the state of $site_1$ becomes "effects". On the contrary, when op_1 is received on $site_2$, the transformation does not modify op_1's parameters since f is inserted before s. Thus, op_1 is executed as-is and the state of $site_2$ is "effects". On this scenario, OT approach has ensured that both copies converge to the same value.

The OT approach distinguishes two main components: an *integration algorithm* and a set of *transformation functions*. The integration algorithm is in charge of reception, diffusion and execution of operations. When necessary, it calls transformation functions. This algorithm does not depend on type of replicated data. The transformation functions merge concurrent modifications by serializing two concurrent operations. These functions are specific to a particular type of replicated data such as string of characters, XML documents, calendars or file system.

OT approach aims to achieve *convergence* of copies.

a) Convergence: As every optimistic replication algorithm, OT approach aims to ensure eventual consistency. This means that if no updates are performed for a long period of time, all updates will eventually propagate through the system and all the copies will converge towards a same value. In other words, when the system is idle (no operation in pipes), all copies are identical.

To ensure convergence, it has been proved [15] that the underlying transformation functions must satisfy two properties:

Definition The TP_1 property defines a *state equivalence*. The state generated by the execution of op_1 followed by $T(op_2, op_1)$ must be the same as the state generated by the execution of op_2 followed by $T(op_1, op_2)$: $op_1 \circ T(op_2, op_1) \equiv op_2 \circ T(op_1, op_2)$

Definition The TP_2 property ensures that the transformation of an operation regarding a sequence of concurrent operations does not depend on the order in which operations of this sequence were transformed: $T(op_3, op_1 \circ T(op_2, op_1)) = T(op_3, op_2 \circ T(op_1, op_2))$

The operational transformation approach could be used to design a reconciliation framework able to reconciliate divergent copies of any type of data. In order to build such a framework, the following task have to be completed. First, an integration algorithm must be chosen ; regarding this algorithm, TP_2 property may be required on underlying transformation functions. Second, operations which could be

performed on shared data types must be defined. Finally, the required transformation functions for all combination of operations have to be provided.

III. Motivating Example

Now imagine that three authors ($UserU$, $UserV$ and $UserW$) working together to modify the following XML document.

```
<?xml version="1.0" encoding="UTF-8"?>
<EXAMPLE>
  <DATA>
    <TITLE>T1</TITLE>
    <B>B1</B>
 </DATA>
<DATA>
    <TITLE>T2</TITLE>
    <A>A2</A>
    <B>B2</B>
    <C>C2</C>
 </DATA>
</EXAMPLE>
```

This document has the following constraints (DTD):

- C1 (Uniqueness): The element *TITLE* must be unique.
- C2 (Existence): The element *A* must be followed by the element *B* i.e. it is not possible to have *A* without *B* (A;B).
- C3 (Alternative): There is an alternative between element *C* and *D* (C or D and not both of them)

Each author has his own replica of the initial document. The author $UserU$ starts to modify his copy. We simplified the operation profiles in order to improve readability. $UserU$ produces the following sequence of operations:

1) $op_1 = update(C2, CU2);$. The update operation is interpreted as $delete(C1)$ followed by $insert(CU2)\, after\, B2$.
2) $op_2 = update(T1, TU1)$
3) $op_3 = Insert(< A > A1 < /A >)\, before\, B1$

$userU$ produces the following valid XML document:

```
<?xml version="1.0" encoding="UTF-8"?>
<EXAMPLE>
  <DATA>
    <TITLE>TU1</TITLE>
    <A>A1</A>
    <B>B1</B>
 </DATA>
<DATA>
    <TITLE>T2</TITLE>
    <A>A2</A>
    <B>B2</B>
    <C>CU2</C>
 </DATA>
</EXAMPLE>
```

Concurrently, the author $UserV$ produces the following operations:

1) $op_4 = update(T1, TV1)$
2) $op_5 = delete(B1)$

So $UserV$ is producing the following valid XML document:

```
<?xml version="1.0" encoding="UTF-8"?>
<EXAMPLE>
  <DATA>
    <TITLE>TV1</TITLE>
 </DATA>
<DATA>
    <TITLE>T2</TITLE>
    <A>A2</A>
    <B>B2</B>
    <C>C2</C>
 </DATA>
</EXAMPLE>
```

At the same time, the third author $UserW$ produces the following operations:

1) $op_6 = delete(C2)$
2) $op_7 = insert(< D > DW2 < /D >)\, after\, B2$

$UserW$ produces the following valid XML document:

```
<?xml version="1.0" encoding="UTF-8"?>
<EXAMPLE>
  <DATA>
    <TITLE>T1</TITLE>
    <B>B1</B>
 </DATA>
<DATA>
    <TITLE>T2</TITLE>
    <A>A2</A>
    <B>B2</B>
    <D>DW2</D>
 </DATA>
</EXAMPLE>
```

In order to integrate the different modifications, we need to apply a reconciliation algorithm. We used the SO6 tool [12] with XML transformation functions. SO6 is the only available merging tool that ensures the convergence of the replicas. By using SO6, all sites eventually converge to the following XML document:

```
<?xml version="1.0" encoding="UTF-8"?>
<EXAMPLE>
  <DATA>
    <TITLE>TU1</TITLE>
    <TITLE>TV1</TITLE>
    <A>A1</A>
 </DATA>
<DATA>
    <TITLE>T2</TITLE>
    <A>A2</A>
    <B>B2</B>
    <C>CU2</C>
    <D>DW2</D>
 </DATA>
</EXAMPLE>
```

The merged document does not respect the DTD:

- It violates the constraint C1. For instance, there are two titles in the first element DATA.
- It violates also the constraint C2, there is an element *A* without *B* .

- Finally, it violates the alternative constraint C3 in the second element DATA.

As we can see, in spite that individual modifications of authors respect the DTD the merged document does not. The resulting document will be considered as corrupted when trying to open it. To overcome this problem, we can imagine three solutions.

1) The first solution is to combine OT with DTD validation. This means, that the merging algorithm has to take into account constraints violation. Therefore, we have to write and prove transformation functions. This means, for example, for a DTD with 20 different elements we have to write 400 transformation functions. This is not realistic.
2) The second solution is to use the approach proposed by IceCube [6]. IceCube tries to find an ordering (Scheduling) that respects the constraints. There are two kind of constraints *Static constraints* and *dynamic constraints*. The first one restricts a priori the possible orderings between two actions. For example, an update operation on an object x must precede a delete operation of that object. Therefore, the order Update before Delete is safe (acceptable), however, delete followed by update is unsafe. A dynamic constraint is a precondition of the operation, it is used to check that the state of the object universe at simulation time is compatible with the one observed during isolated execution. The major drawbacks of IceCube are the risk of combinatorial explosion, the annulation of some operations and finally how to transform *DTD* to IceCube constraints
3) The last solution is to repair violation after the merge. The idea of repairing constraints after violations is largely studied in active database domains [2], [5], [9], [10]. While in traditional database, a transaction that violates constraints is rolled-back. In active database, database management system reacts autonomously to inconsistencies, by triggering a set of repairs actions capable of progressive elimination of constraints violation until a consistent state is reached. Final state can be chosen within a subspace of states as compliant as possible to the original intention of the transaction.

We propose a repairing approach for collaborative writing of XML documents. We believe that repairing approach is a reasonable solution to reestablish the consistency of the merged document.

IV. Automatic repairing

We studied most repairing algorithms in active databases, most of them are not applicable for XML documents, they are adapted for relational data model [13]. Only the work proposed by Xlinkit [9], [10] is applicable.

Xlinkit is a framework for maintaining consistency of distributed XML documents. It defines constraints language based on first order logic with XPath syntax. The framework proposes a validation engine for XML document in addition of repairing algorithm.

In the following sections, we show how we can adapt Xlinkit to our context.

A. DTD constraints

In this section, we show how we can use Xlinkit constraints language to describe the DTD constraints of the collaborative writing example in the section III.

The uniqueness constraint (C1: only one title) can be written as:

$$\forall d \in \text{“}/example/data\text{“}((\exists t \in \text{“}\$d/title\text{“}) \wedge (\forall x \in \text{“}\$d/title\text{“}(\forall y \in \text{“}\$d/title\text{“}(\text{“}\$x = \$y''))))$$

The sequence constraint (C2: A ; B) can be written as:

$$\forall d \in \text{“}/example/data\text{“}((\forall x \in \text{“}\$d/A\text{“}(\exists y \in \text{“}d/B\text{“}))$$

Finally, the alternative constraint(C3: C or D) can be written as:

$$\forall x \in \text{“}/example/data\text{“}((\forall y \in' \$x/C'(\neg(\exists z \in \text{“}\$x/D\text{“}))$$

The constraints language is based on first order logic with XPath syntax in order to select sets of elements [9].

B. Repairing algorithm

The repairing algorithm is a kind of rewriting process [10]. According to the constraint expression, some rewriting rules can be applied. Figure 1 shows some of these rules [10].

We describe these rules by applying them to the motivating example. We examine the violation of the existence constraint. Xlinkit generates an inconsistency link to element $/EXAMPLE/DATA[1]$: there is an A element without B.

The constraint $C2$ is proceeded as follows:

1) function D_i generates delete operation for the inconsistent d; {{ delete 1}} where 1 is the locater number returned by Xlinkit.
2) It then calls D_i recursively to process the universal quantifier :$D_i[\forall x \in \text{“}\$d/A\text{“}]$.
3) D_i generates delete operation {{ delete Direct locater(x)}}. And finally the existence quantifier is proceed. This generates the action: {{ add Lookup $\$d/B$ Direct locater(x)}}.

The algorithm proposes three possible repairing actions (cf figure**??**).

```
>>> LINK INCONSISTENT   :: rules\Rule.xml#/consistencyruleset/consistencyrule[3]
    >>> [DONNEES]...[files\Document.xml]...[1]...[/EXEMPLE/DONNEES[1]]
    >>> [A]...[files\Document.xml]...[2]...[/EXEMPLE/DONNEES[1]/A[1]]
    Actions de réparation.....................
        Delete files\Document.xml#/EXEMPLE/DONNEES[1]
        Delete files\Document.xml#/EXEMPLE/DONNEES[1]/A[1]
        Insert files\Document.xml#/EXEMPLE/DONNEES[1]/B
```

Fig. 2. Possible repairing actions

1) {{ delete 1}} : this repairing action implies the deletion of the node located at $/EXAMPLE/DATA[1]$.
2) {{ delete Direct locater(x)}} : this repairing action implies the deletion of the element A which is not followed by B.

$$
\begin{array}{rcl}
\mathcal{D}_i & : & formula \rightarrow \wp(\wp(Action)) \\
\mathcal{D}_i[\![\forall \mathbf{var} \in XPath\ (formula)]\!]_{\sigma,\tau} & = & \{\{Delete\ Direct\ locator(\mathbf{var})\}\} \times \\
 & & \mathcal{D}_i[\![formula]\!]_{(\{(var,XPath)\}\cup\sigma),(\{var\}\cup\tau)} \\
\mathcal{D}_i[\![\exists \mathbf{var} \in XPath\ (formula)]\!]_{\sigma,\tau} & = & \{\{Add\ Lookup\ XPath\}\} \times \mathcal{N}_i[\![formula]\!]_{\{(var,XPath)\}\cup\sigma,\tau} \\
\mathcal{D}_i[\![formula_1\ \mathbf{and}\ formula_2]\!]_{\sigma,\tau} & = & \mathcal{D}_i[\![formula_1]\!]_{\sigma,\tau} \cup \mathcal{D}_i[\![formula_2]\!]_{\sigma,\tau} \cup (\mathcal{D}_i[\![formula_1]\!]_{\sigma,\tau} \times \mathcal{D}_i[\![formula_2]\!]_{\sigma,\tau}) \\
\mathcal{D}_i[\![formula_1\ \mathbf{or}\ formula_2]\!]_{\sigma,\tau} & = & \mathcal{D}_i[\![formula_1]\!]_{\sigma,\tau} \times \mathcal{D}_i[\![formula_2]\!]_{\sigma,\tau} \\
\mathcal{D}_i[\![formula_1\ \mathbf{implies}\ formula_2]\!]_{\sigma,\tau} & = & \mathcal{D}_c[\![formula_1]\!]_{\sigma,\tau} \times \mathcal{D}_i[\![formula_2]\!]_{\sigma,\tau} \\
\mathcal{D}_i[\![\mathbf{not}\ formula]\!]_{\sigma,\tau} & = & \mathcal{D}_c[\![formula]\!]_{\sigma,\tau} \\
\mathcal{D}_i[\![RelativePath_1 = RelativePath_2]\!]_{\sigma,\tau} & = & change(RelativePath_1,\sigma,\tau) \times change(RelativePath_2,\sigma,\tau)
\end{array}
$$

$$
\begin{array}{rcl}
\mathcal{D}_c & : & formula \rightarrow \wp(\wp(Action)) \\
\mathcal{D}_c[\![\forall \mathbf{var} \in XPath\ (formula)]\!]_{\sigma,\tau} & = & \{\{Add\ Lookup\ XPath\}\} \times \mathcal{N}_i[\![formula]\!]_{\{(var,XPath)\}\cup\sigma,\tau} \\
\mathcal{D}_c[\![\exists \mathbf{var} \in XPath\ (formula)]\!]_{\sigma,\tau} & = & \{\{Delete\ Direct\ locator(\mathbf{var})\}\} \times \\
 & & \mathcal{D}_c[\![formula]\!]_{(\{(var,XPath)\}\cup\sigma),(var\cup\tau)} \\
\mathcal{D}_c[\![formula_1\ \mathbf{and}\ formula_2]\!]_{\sigma,\tau} & = & \mathcal{D}_c[\![formula_1]\!]_{\sigma,\tau} \times \mathcal{D}_c[\![formula_2]\!]_{\sigma,\tau} \\
\mathcal{D}_c[\![formula_1\ \mathbf{or}\ formula_2]\!]_{\sigma,\tau} & = & \mathcal{D}_c[\![formula_1]\!]_{\sigma,\tau} \cup \mathcal{D}_c[\![formula_2]\!]_{\sigma,\tau} \cup (\mathcal{D}_c[\![formula_1]\!]_{\sigma,\tau} \times \mathcal{D}_c[\![formula_2]\!]_{\sigma,\tau}) \\
\mathcal{D}_c[\![formula_1\ \mathbf{implies}\ formula_2]\!]_{\sigma,\tau} & = & \mathcal{D}_i[\![formula_1]\!]_{\sigma,\tau} \cup \mathcal{D}_c[\![formula_2]\!]_{\sigma,\tau} \\
\mathcal{D}_c[\![\mathbf{not}\ formula]\!]_{\sigma,\tau} & = & \mathcal{D}_i[\![formula]\!]_{\sigma,\tau} \\
\mathcal{D}_c[\![RelativePath_1 = RelativePath_2]\!]_{\sigma,\tau} & = & change(RelativePath_1,\sigma,\tau) \times change(RelativePath_2,\sigma,\tau)
\end{array}
$$

Fig. 1. Rules for repairing

3) {{ add Lookup $\$d/B$ Direct locater(x)}} : this repairing action implies the insertion of an element B to complete the element A.

As we can see the algorithm proposes a set of repairing actions and it is up to the user to choose one. In the context of collaborative writing, the first solution is not acceptable since it implies the deletion of the whole element i.e. lost of many data. The second solution proposes to locate the deletion to the only concerned element i.e. deletes the element A for which there is no B. Again with this solution there is a lost of data. The final solution proposes to add a new element, this is the best one since there is no loose of data.

Xlinkit proposes a set of repairing actions. The user has to choose one. In collaborative writing this not an acceptable solution since different users can choose different repairing actions. For example, $UserU$ can choose the second repairing action while $UserW$ chooses the third one. In this case, there is a divergence, the synchronization algorithm is restarted, then the repairing algorithm restarted. There is an infinite loop of synchronization and repairing cycles.

A possible solution to this problem is to let the system chooses a repairing action in a deterministic way. This allows to ensures that all sites will apply the same repairing actions and therefore they will converge toward the same valid XML document.

We developed a system that set priorities to repairing actions. Each action is decorated with a priority. Action with priority 1 is preferable to action with priority -1. We attribute the highest priority to the Insertion action. It has a priority 1. A deletion action has a priority < 1. To establish an order among deletion actions, we propose to use the size of the deleted data. If we apply priorities rules to the previous example, the system will generates:

1) {{ delete 1 priority -1 }}
2) {{ delete Direct locater(x) priority 0 }}
3) {{ add Lookup $\$d/B$ Direct locater(x) priority 1 }}

The third action has the highest priority. Therefore, it will be chosen as a repairing action. All sites will have an element B with a default value as shown below.

```
<?xml version="1.0" encoding="UTF-8"?>
<EXAMPLE>
  <DATA>
    <TITLE>TU1</TITLE>
    <TITLE>TV1</TITLE>
    <A>A1</A>
    <B>DEFAULT_B</B>
 </DATA>
<DATA>
    <TITLE>T2</TITLE>
    <A>A2</A>
    <B>B2</B>
    <C>CU2</C>
    <D>DW2</D>
 </DATA>
</EXAMPLE>
```

If this reparation is not very impressive, it allows to open the document with the tool that produced it. At the same time, it

respects the requirements of collaborative writing by avoiding lost update.

In the same way, the function D_i is recursively applied to repair uniqueness and alternative constraints.

V. Conclusion

Collaborative writing is the process by which more than one author share opinions and contribute to the content of a document. Collaborative writing is standard practice in technical and scientific settings; some examples include research papers, software development, proposals for funding and user manuals.

In this paper, we address the collaborative writing of XML document. While some XML reconciliation algorithms ensure convergence, they do not ensure any guarantee about the validation of the DTD.

We propose an automatic repairing approach to reestablish consistency. We show the proposed solution through an example. This solution is a first step to handle inconsistency of merged XML document. We validated the approach through an implementation.

As a perspective of this work, we plan to use the group 'undo' operation [18] as repairing action. Currently, repairing algorithms only consider insert, delete or update operation. We believe that a group undo can improve significantly the quality of reparation.

References

[1] DeltaXML, managing change in an xml environment. http://www.deltaxml.com/.

[2] S. Ceri, P. Fraternali, S. Paraboschi, and L. Tanca. Automatic generation of production rules for integrity maintenance. *ACM Trans. Database Syst.*, 19(3):367–422, 1994.

[3] P. Dourish. The Parting of the Ways: Divergence, Data Management and Collaborative Work. In *Proceedings of the European Conference on Computer-Supported Cooperative Work - ECSCW'95*, pages 215–230, Stockholm, Sweden, September 1995. Kluwer Academic Publishers.

[4] C. A. Ellis and S. J. Gibbs. Concurrency Control in Groupware Systems. 18:399–407, May 1989.

[5] M. Gertz and U. W. Lipeck. An extensible framework for repairing constraint violations. In *Proceedings of the IFIP TC11 Working Group 11.5, First Working Conference on Integrity and Internal Control in Information Systems*, pages 89–111, London, UK, UK, 1997. Chapman & Hall, Ltd.

[6] A.-M. Kermarrec, A. Rowstron, M. Shapiro, and P. Druschel. The IceCube Approach to the Reconciliation of Divergent Replicas. In *Proceedings of the ACM Symposium on Principles of Distributed Computing - PODC 2001*, pages 210–218, Newport, Rhode Island, USA, August 2001. ACM Press.

[7] P. Molli, G. Oster, H. Skaf-Molli, and A. Imine. Using the Transformational Approach to Build a Safe and Generic Data Synchronizer. In *Proceedings of the ACM SIGGROUP Conference on Supporting Group Work - GROUP 2003*, pages 212–220, Sanibel Island, Florida, USA, November 2003. ACM Press.

[8] P. Molli, H. Skaf-Molli, G. Oster, and S. Jourdain. SAMS: Synchronous, Asynchronous, Multi-Synchronous Environments. In *Proceedings of the Conference on Computer-Supported Cooperative Work in Design - CSCWD 2002*, pages 80–85, Rio de Janeiro, Brazil, September 2002.

[9] C. Nentwich, L. Capra, W. Emmerich, and A. Finkelstein. xlinkit: a consistency checking and smart link generation service. *ACM Trans. Inter. Tech.*, 2(2):151–185, 2002.

[10] C. Nentwich, W. Emmerich, and A. Finkelstein. Consistency management with repair actions. In *ICSE '03: Proceedings of the 25th International Conference on Software Engineering*, pages 455–464, Washington, DC, USA, 2003. IEEE Computer Society.

[11] S. Noël and J.-M. Robert. Empirical study on collaborative writing: What do co-authors do, use, and like? *Computer Supported Cooperative Work - JCSCW*, 13(1):63–89, 2004.

[12] G. Oster, H. Skaf-Molli, P. Molli, and H. Naja-Jazzar. Supporting Collaborative Writing of XML Documents. In *Proceedings of the International Conference on Enterprise Information Systems - ICEIS 2007*, Funchal, Madeira, Portugal, jun 2007.

[13] D. RAZAFIMAHATRATRA. Rparation automatique dans un environnement collaboratif. Master's thesis, LORIA-INRIA Lorraine, University Henri Poincar, Nancy1, 2004.

[14] H. Skaf-Molli, P. Molli, and G. Oster. Semantic Consistency for Collaborative Systems. In *Proceedings of the International Workshop on Collaborative Editing Systems - CEW 2003*, Helsinki, Finlande, September 2003.

[15] M. Suleiman, M. Cart, and J. Ferrié. Concurrent Operations in a Distributed and Mobile Collaborative Environment. In *Proceedings of the International Conference on Data Engineering - ICDE'98*, pages 36–45, Orlando, Florida, USA, February 1998. IEEE Computer Society.

[16] C. Sun and D. Chen. Consistency Maintenance in Real-Time Collaborative Graphics Editing Systems. *ACM Transactions on Computer-Human Interaction*, 9(1):1–41, March 2002.

[17] S. G. Tammaro, J. N. Mosier, N. C. Goodwin, and G. Spitz. Collaborative Writing Is Hard to Support: A Field Study of Collaborative Writing. *Computer-Supported Cooperative Work - JCSCW*, 6(1):19–51, 1997.

[18] S. Weiss, P. Urso, and P. Molli. Compensation in collaborative editing. In *International Workshop on Collaborative Editing Systems*, Sanibel Island, USA, November 2007.